METHOD FOR EXPLICIT DATA RATE CONTROL IN A PACKET COMMUNICATION ENVIRONMENT WITHOUT DATA RATE SUPERVISION

CROSS-REFERENCES TO RELATED APPLICATIONS

The present application is a division application of U.S. patent application Ser. No. 08/742,994 which was filed on Nov. 1, 1996 now U.S. Pat. No. 6,038,216 naming Robert L. Packer as inventor and is incorporated by reference herein in its entirety.

COPYRIGHT NOTICE

BACKGROUND OF THE INVENTION

This invention relates to digital packet telecommunications, and particularly to data flow rate control at a particular layer of a digitally-switched packet telecommunications environment normally not subject to data flow rate control wherein data packets are communicated at a variety of rates without supervision as to rate of data transfer, such as under the TCP/IP protocol suite.

The widely-used TCP/IP protocol suite, which implements the world-wide data communication network environment called the Internet and is employed in local networks also (Intranets), intentionally omits any explicit supervisory function over the rate of data transport over the various media which comprise the network. While there are certain perceived advantages, this characteristic has the consequence of juxtaposing very high-speed packets and very low-speed packets in potential conflict and the resultant inefficiencies. Certain loading conditions can even cause instabilities which could lead to overloads that could stop data transfer temporarily. It is therefore considered desirable to provide some mechanism to optimize efficiency of data transfer while minimizing the risk of data loss.

In order to understand the exact context of the invention, an explanation of technical aspects of the Internet/Intranet telecommunications environment may prove helpful.

Internet/Intranet technology is based largely on the TCP/IP protocol suite, where IP is the network level Internet Protocol and TCP is the transport level Transmission Control Protocol. At the network level, IP provides a "datagram" delivery service. By contrast, TCP builds a transport level service on top of the datagram service to provide guaranteed, sequential delivery of a byte stream between two IP hosts.

TCP has `flow control` mechanisms operative at the end stations only to limit the rate at which a TCP endpoint will emit data, but it does not employ explicit data rate control. The basic flow control mechanism is a `sliding window`, a time slot within an allowable window which by its sliding operation essentially limits the amount of unacknowledged transmit data that a transmitter can emit.

Another flow control mechanism is a congestion window, which is a refinement of the sliding window scheme involving a conservative expansion to make use of the full, allowable window. A component of this mechanism is sometimes referred to as `slow start`.

The sliding window flow control mechanism works in conjunction with the Retransmit Timeout Mechanism (RTO), which is a timeout to prompt a retransmission of unacknowledged data. The timeout length is based on a running average of the Round Trip Time (RTT) for acknowledgment receipt, i.e. if an acknowledgment is not received within (typically) the smoothed RTT+4*mean deviation, then packet loss is inferred and the data pending acknowledgment is retransmitted.

Data rate flow control mechanisms which are operative end-to-end without explicit data rate control draw a strong inference of congestion from packet loss (inferred, typically, by RTO). TCP end systems, for example, will `back-off`, i.e., inhibit transmission in increasing multiples of the base RTT average as a reaction to consecutive packet loss.

Bandwidth Management in TCP/IP Networks

Bandwidth management in TCP/IP networks is accomplished by a combination of TCP end systems and routers which queue packets and discard packets when some congestion threshold is exceeded. The discarded and therefore unacknowledged packet serves as a feedback mechanism to the TCP transmitter. (TCP end systems are clients or servers running the TCP transport protocol, typically as part of their operating system.)

The term `bandwidth management` is often used to refer to link level bandwidth management, e.g. multiple line support for Point to Point Protocol (PPP). Link

level bandwidth management is essentially the process of keeping track of all traffic and deciding whether an additional dial line or ISDN channel should be opened or an extraneous one closed. The field of this invention is concerned with network level bandwidth management, i.e. policies to assign available bandwidth from a single logical link to network flows.

Routers support various queuing options. These options are generally intended to promote fairness and to provide a rough ability to partition and prioritize separate classes of traffic. Configuring these queuing options with any precision or without side effects is in fact very difficult, and in some cases, not possible. Seemingly simple things, such as the length of the queue, have a profound effect on traffic characteristics. Discarding packets as a feedback mechanism to TCP end systems may cause large, uneven delays perceptible to interactive users.

Routers can only control outbound traffic. A 5% load or less on outbound traffic can correspond to a 100% load on inbound traffic, due to the typical imbalance between an outbound stream of acknowledgments and an inbound stream of data.

A mechanism is needed to control traffic which is more efficient in that it is more tolerant of and responsive to traffic loading.

As background, further information about TCP/IP and the state of the art of flow control may be had in the following publications:

Comer, Douglas. Internetworking with TCP/IP. Vol. I. Prentice Hall, 1991.

Comer, Douglas and Stevens, David. Internetworking with TCP/IP. Vol. II. Design, Implementation, and Internals. Prentice Hall, 1991.

W. Richard Stevens, TCP/IP Illustrated. Vol. I--The Protocols. Addison-Wesley. 1994.

RFC 793. Transmission Control Protocol. Postel, 1981.

RFC 1122. Host Requirements. Braden 1989.

A particularly relevant reference to the present work is:

Hari Balakrishnan, Srinivasan Seshan, Elan Amir, Randy H. Katz. Improving TCP/IP Performance over Wireless Networks. Proc. 1 st ACM Conf. on Mobile Computing and Networking, Berkeley, Calif., November 1995.

The above document reports efforts of a research group at the University of California at Berkeley to implement TCP `interior spoofing` to mitigate the effects of single

packet loss in micro-cellular wireless networks. Its mechanism is the buffering of data and performing retransmissions to preempt end to end RTO. It is a software mechanism at a wireless network based station which will aggressively retry transmission a single time when it infers that a single packet loss has occurred. This is a more aggressive retransmission than the normal TCP RTO mechanism or the `quick recovery` mechanism, whereby a transmitter retransmits after receiving N consecutive identical acknowledgments when there is a window of data pending acknowledgment.

Sliding window protocols are known, as in Comer, Vol. I: page 175. Known sliding window protocols are not time based. Rate is a byproduct of the characteristics of the network and the end systems.

SUMMARY OF THE INVENTION

According to the invention, a method for explicit network level data rate control is introduced into a level of a packet communication environment at which there is a lack of data rate supervision to control assignment of available bandwidth from a single logical link to network flows. The method includes adding latency to the acknowledgment (ACK) packet of the network level and adjusting the reported size of the existing flow control window associated with the packet in order to directly control the data rate of the source data at the station originating the packet.

Called direct feedback rate control, the method comprises a mechanism that mitigates TCP packet level traffic through a given link in order to manage the bandwidth of that link. A software mechanism to implement the function may be a software driver, part of a kernel of an operating system or a management function implemented on a separate dedicated machine in the communication path.

The invention has the advantage of being transparent to all other protocol entities in a TCP/IP network environment, For example, in the connections controlled according to the invention, it is transparent to TCP end systems (i.e., end systems using the TCP protocol).

The invention will be better understood upon reference to the following detailed description in connection with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 is a block diagram of a system according to the invention.

FIGS. 2A-2I depict flowcharts of a particular embodiment according to the invention.

DESCRIPTION OF THE SPECIFIC EMBODIMENTS

Referring to FIG. 1, a system 10 which uses the invention comprises a first TCP end system 12, such as a server device, a second TCP end system 14, such as a client device, which is connected through a first router 16 and thence via a first access link 18 into a network cloud 20 of a character as hereinafter explained, which in turn provides a logical connection via a second access link 22 to a second router 24. According to the invention, there is provided between at least one of the end systems 12 and one of the routers 24 a rate control device 26 which is operative to control the rate at which a TCP transmitter can emit packets. In this invention, rate control is accomplished by (1) delaying the transmission of an acknowledgment, which may be a special packet, (known as an ACK) issued by the end system 12 or 14; and/or by (2) modifying the report of the length of the receive window in those same packets; and/or by (2) generating acknowledgments substantially independently of the acknowledgments generated by the end receiving station. The rate control device 26 is preferably placed at the end system acting as the server, but a rate control device may be placed adjacent any system in a path for data. At the server, it is most effective in controlling flow rate because it typically receives and relays the bulk of the traffic of interest.

Several factors are employed to weight the length of the added acknowledgment delay: the amount of data being acknowledged, the measured and smoothed round trip time, the targeted transmission rate, and the mean of the deviation between the measured and smoothed roundtrip time.

The direct feedback rate control mechanism may be incorporated conveniently into computer-executable code, assuming the end system or the router is programmable. It can be used in an open loop environment or it can be used with a rate monitoring mechanism that provides an indication of flow rate.

The following is a detailed description of a specific embodiment of a direct feedback rate control mechanism expressed in pseudocode. For example, the pseudo-code explains a mechanism of generating acknowledgments, including substituting acknowledgments. This pseudo-code also addresses specifics of the TCP environment. It

should be noted that latency and the sliding window flow control can be implemented anywhere in the data path.

According to an embodiment of the present invention, a method for controlling data rate of data packets in a digital data packet communication environment is provided. This environment employs TCP/IP protocols and has a plurality of interconnected digital packet transmission stations. TCP/IP data packet communication environments lack explicit end-to-end data rate supervision, such as that found in ATM or Frame Relay systems. In this invention, a first digital packet transmission station sends at least one source packet to a second digital packet transmission station over the network. The first digital packet transmission station waits for typically one acknowledgment packet before sending additional source packets. The second digital packet transmission station sends an acknowledgment packet toward the first digital packet transmission station after receipt of the at least one source packet.

The second digital packet transmission station can thus establish a limit on explicit rate of emission of packets from the first transmission station to the second transmission station. It invokes this limit by the timing of transmission of the acknowledgment packet from the second digital packet transmission station to the first digital packet transmission station. Specifically, the second digital packet transmission station will insert a latency or delay in issuance of the acknowledgment packet. The latency will be based upon the selected limit. The latency can be invoked at any point traversed by the acknowledgment packet by delaying propagation of the acknowledgment packet through that point. One way this is done is by intercepting the acknowledgment packet at an intermediate point, and rescheduling it and reusing it, i.e., substituting a duplicate acknowledgment packet at a delay from normal propagation. An alternative to the duplicate substitution is the substitution of a sequence of values which include the acknowledgment sequence indicators. The substitution sequence might also include other indicia, for example, an indication of a suitable window size to be communicated to the first transmission station. These indicators serve to signal to the first transmission station the information needed to repackage groupings of packets, which also imposes an upper limit and explicit transmission rate.

Substitute acknowledgment packets can be generated at any point of latency in response to groupings of packets from the first digital packet transmission station. The acknowledgments and the groupings can be generated without correlation to time, range of

data being acknowledged, window size being advertised to the first transmission station and number of acknowledgments. The selection of the latency time in theory can be a manually preset value. Alternatively it can be selected based on a simple formula relating latency to explicit maximum rate of transmission. Since the rate of emission of packets from a source or first transmission station to the second transmission station is a limited by the fact that acknowledgments must be received before the next emission of a packet or grouping of packets, the latency of acknowledgment imposes an upper bound on explicit transmission rate.

FIG. 2A depicts a flow chart 201 of processing steps in the subroutine tcpSchedule. This routine is invoked for every TCP segment to schedule its emission. In a decisional step 202, a check is made whether there is TCP data present. If this is so, then in a step 204 the data rate received for this connection is updated and processed. Otherwise, or in any event, processing continues with a decisional step 206 wherein a check is made whether the packets being tested are the start of a new burst. If this is so, then in a step 208, an appropriate policy for this connection and this direction of traffic flow is found. This yields a combination of CIR (committed information rate) and EIR (excess information rate). Otherwise, or in any event, processing continues with a decisional step 210 wherein a check is performed to see if there is a target rate associated with this flow. If there is no target rate, then in a step 212, the packet is scheduled for an immediate transmission after which processing returns. Otherwise, if there is a target rate for this particular flow, then in a decisional step 214, a check is performed to see if an acknowledgment packet or sequence (ACK) for the outbound retransmission of TCP data is being retained. This check determines whether: 1) there is TCP data AND 2) it is a retransmitted data packet and 3) an ACK is being retained for this data. If all these conditions are true, the retransmitted data packet is discarded in a step 216. A buffer containing the packet is cleared and processing returns. This step ensures that there is no retransmission caused by our own acknowledgment delay. If this is not so, then in a step 218, a check is made to see if this acknowledgment (ACK) sequence is greater than the last forwarded acknowledgment sequence. If this is so, then in a decisional step 220, a check is made to see if the data rate on the other half connection (data flowing in the other direction along this connection) exceeds its target rate.

FIG. 2B depicts a continuation flow chart 203. If the test in step 218 in FIG. 2A of whether this ACK sequence is greater than the last forwarded ACK sequence fails,

then processing continues with a step 234 in FIG. 2B, wherein the packet is scheduled for immediate transmission and then processing returns. If the test of decisional step 220 in FIG. 2A, of whether the data rate on the other half connection exceeds its target rate, fails, then processing continues with a step 222 in FIG. 2B, wherein a last ACK forwarded (last acknowledgment forwarded) variable is set for this particular half connection to tcphdr pointing on ACK. Then in a step 224, variable timeLastFreshAckForwarded for this half connection is set to the current time in order to cause an immediate transmit. Then in step 234, the packet is scheduled for immediate transmission and processing returns. Finally, if the test of step 220 in FIG. 2A, whether the data rate on the other half connection exceeds its target rate, is true, then processing continues with a decisional step 226 in FIG. 2B, in which a check is made whether there is TCP data to schedule. If this is not so, then in a step 228, a procedure tcpSeparateData is invoked in order to change the ACK value on the data packet and forward it immediately. Then in a decisional step 230, a test is performed to determine whether a new ACK was created. If step 230 is not true, then the processing returns. Otherwise a new ACK was created, so processing continues with a step 232, in which routine tcpDownShift is invoked to handle TCP ACK packets. After step 232, processing returns. If the test in decisional step 226 fails because there is no TCP data to schedule, then processing continues directly with step 232 to invoke tcpDownShift to handle TCP packets. After step 232, processing returns.

FIG. 2C depicts a flow chart 205 of the process steps of routine tcpSeparateData which is step 228 of FIG. 2B. This subroutine is invoked by routine tcpSchedule to separate data from IP and TCP headers, assign it an older (i.e., prior) ACK and schedule the transmission, or to create a new ACK-only packet with specified ACK sequence and to return it. In a decisional step 240, a determination is made whether an ACK packet for this connection already exists. If this is not so, then a new ACK packet is created. In a step 242, the front of the packet, i.e. the IP and TCP headers, is copied into the newly created ACK packet. Then in a step 244, the length in the newly created ACK packet is updated in order to exclude the length of the data.

If the test in step 240 is true, then in a step 246, a flag for this half connection is set to cause a reuse of the existing ACK packet when it is eventually recycled at transmit complete time. Subsequently, in a step 248, the ACK and checksum in the original packet are altered and it is sent. Then in a decisional step 250, a determination is made whether a new

ACK packet was created. If this is so, then in a step 252 the checksum is computed and set for the newly created packet. Otherwise, or in any event, the ACK packet is returned.

FIG. 2D depicts a flow chart 207 of the processing steps of subroutine tcpDownShift. This subroutine is called from function tcpSchedule and from a recycle routine to schedule an acknowledgment for a half connection which is exceeding its target rate based on information supplied to it. TcpDownShift is the function to choose a latency time to establish a limit on explicit rate of emission of packets from the first transmission station to the second transmission station. One technique for establishing the limit on rate is to generate substitute acknowledgment packets at a point of latency in response to groupings of packets received from the first digital packet transmission station. In a first decisional step 255, a determination is made whether there is already an acknowledgment outstanding for this half connection. If this is so, then in a step 257, that outstanding acknowledgment packet is marked for reuse at the appropriate time after its scheduled transmission and the routine returns to its caller. Otherwise, in a step 256, a new acknowledgment packet is prepared. An indicator for this half connection's acknowledgment packet is set to point to the new acknowledgment packet. A recycle function and argument to capture the transmit complete confirmation from the driver is also associated with the new acknowledgment packet. Then, in a first decisional step 260, a determination is made whether the amount of data to be acknowledged is greater than two times the MSS (Maximum Segment Size). If this is the case, then in a step 262, processing "clamps" the new acknowledgment at the lastAcknowledgementForwarded added to the amount of two times the MSS. This has the effect of limiting the acknowledgment to the position where the last acknowledgment occurred, plus a number of bytes equal to two times the MSS. Otherwise, or in any event, in a step 263, values are computed for: 1) a time since the last acknowledgment was sent for the particular connection, 2) a target data rate for the connection, 3) an actual rate for the connection and 4) an amount of time until the acknowledgment must be sent back to achieve the target data rate. The time to send an acknowledgment is computed from the amount of data sent from the first transmission station and the target rate for the connection. Thus, the timing of the acknowledgment is not dependent upon time, range of data being acknowledged, advertised window size and number of acknowledgments. In a decisional step 264, a determination is made whether data rate reduction can be achieved in a single step without incurring an RTO at the sender. In other words, the question is whether the target

rate can be achieved by scheduling just one acknowledgment. If this is so, then in a step 266, the half connection's actual rate is set to the target rate. Otherwise in a step 268, the other half connection's actual rate is set to a ratio of the data rate to the calculated RTO time. Processing then continues with a decisional step 270, in which a determination is made whether the sending of an ACK is late. If this condition is detected, TcpDownShift will try to compensate by acknowledging more data in order to bring the actual data rate closer to the target data rate for this connection. In a decisional step 272, a determination is made whether there is further data which is unacknowledged. If this is the case, then in a step 274, time will be recovered by acknowledging a greater amount of data. In a step 279, a flag is set to indicate a late carryover. Then, in a step 280, the window size is checked to determine that it is not so small as to be causing a late acknowledgment. If the test of decisional step 272 fails, then in a decisional step 276, a determination is made whether there is existing late carryover time for this connection. If a previous carry over exists from a prior late condition, then the current late carry over is adjusted to compensate for the prior late carry over in step 278. Next, TcpDownShift selects the latency for transmitting the acknowledgment that will produce the target data rate on the connection. This latency, stored in variable `delay`, is the greater of the difference between the time until acknowledging, computed above, and zero. This processing is described in a step 281 of FIG. 2E, in which a delay is computed from the difference of the time until the next acknowledgment must be sent to maintain the target data rate and the time since the last acknowledgment was sent. Both of these were computed in step 263 of FIG. 2D. Next, TcpDownShift invokes routine TcpUpdateAckRxWindow in a step 282 to determine whether the advertised receive window is set too large. Processing for TcpUpdateAckRxWindow is described further in FIG. 2H. If this is so, then in step 284 the advertised receive window is scaled down. Otherwise, or in any event, in a step 286, the checksum in the packet header is updated. In a step 288, the acknowledgment packet is scheduled for transmission at a time equal to the present time added to the delay computed in step 281. This processing has chosen a latency time to establish a limit on explicit rate of emission of packets from the first station to the second transmission station. After this processing, control returns to routine TcpSchedule. Then processing returns to the point from which the tcpDownShift was called.

FIG. 2F depicts a flowchart 209 of the process steps of routine tcpRtoTime which is step 264 of FIG. 2B. This subroutine is invoked to return the amount of time until an

RTO will occur, minus the amount of time needed to send an ACK to the source of the RTO, minus a safety margin to account for any noise. The time until RTO is derived from maintaining the smoothed round trip time and a mean deviation and is essentially two times the mean deviation, plus the smoothed round trip time. This is intended to be a conservative value. Many implementations may actually set RTO to a greater value. In a step 290, the RTO time is determined. Then in a step 292, the RTO time determined in step 290 is reduced by the time to send an ACK. Finally, in a step 294, the RTO time reduced by the time to send an ACK determined in step 292, is further reduced by a safety factor. Then processing returns.

FIG. 2G depicts a flowchart 211 of the process steps of routine tcpCheckMinWinSize which is step 280 of FIG. 2D. This subroutine is invoked by routine tcpDownShift whenever there has been a late ACK in order to see if it is desirable to open up the minimum window size. In a step 300, the time since the last increase in window size is determined. Then, in a step 302, the minimum window increase time is multiplied by the maximum window adjustment. Then, in a step 304, the lesser of the quantities computed in steps 300 and 302 is selected. Then, in a step 306, the quantity computed in step 304 is scaled by the minimum window size. Then, in a step 308, the maximum window adjustment in MSS sized segments is reduced by the quantity computed in step 306. Finally, in a step 310, the window is increased by the quantity of MSS sized segments computed in step 308. Thereupon, processing returns.

FIG. 2H depicts a flowchart 213 of the process steps of routine tcpUpdateAckRxWindow, which is step 282 of FIG. 2E. The routine TcpUpdateAckRxWindow selects a substitute indicator for the TCP/IP advertised window size. The advertised receive window is TCP/IP's indicator of window size. Basically, it indicates to the other computers on the network the amount of information this computer can receive at any one time. TcpUpdateAckRxWindow returns Boolean true if the window has been updated, otherwise it returns Boolean false. In decisional step 311, the original amount acknowledged is compared to two times the MSS and a determination is made whether no carryover has occurred. If the original amount acknowledged is greater than two times the MSS, and no carry over has occurred, then in a step 312 the WIN closing flag is checked to determine whether it is not set. The WIN closing flag, when set, causes an adjustment to be made in the next acknowledgment rather than this acknowledgment. If the WIN closing flag

is not set, then in a step 314, the WIN closing flag is set to cause an adjustment to be made in the next ACK and false is returned. Otherwise, if the WIN closing flag has been set, then in a step 316, the window size is set to allow the specified rate given the current round trip time ("RTT") and the detected speed. Then, in a step 318, the receive window is clamped to at least two times the MSS. Then in a step 320, the revised window is moved into the packet and processing returns a true. If, the original amount acknowledged is less than two times the MSS, or there has been a carryover, then in a step 322 the WIN closing flag is reset and a false is returned. This processing performs the function of selecting a substitute indicator for window size to establish a limit on explicit rate of emission of packets from the first transmission station to the second transmission station.

FIG. 2I depicts a flowchart 215 of the process steps of routine tcpSchedAckEmitted, which is invoked whenever a half connection has had its rate reduced in order to recycle the ackpacket for the half connection. This routine is invoked as an indication that a scheduled acknowledgment packet has been sent. But whenever an acknowledgment packet is scheduled, its sequencing information must be checked to insure that acknowledgments are emitted in the right order. In a decisional step 330, a determination is made whether the half connection is still valid. If this is not so, then processing returns. Otherwise, in a step 332, a lastACKforwarded variable is set for this particular half connection. Then in a step 334, variable timeLastFreshAckForwarded for this half connection is set to the current time in order to cause an immediate transmit. Next, decisional step 336 determines if the last acknowledgment sent corresponds to the last acknowledgment received. If this condition is met, no re-sequencing occurs and processing returns to the caller. Otherwise, in a step 338 substitute sequencing values are determined by setting the new acknowledgment to the last acknowledgment received and setting the TCP data length to zero. Then, in step 340, the next acknowledgment time is computed and routine TcpDownShift is invoked to schedule the packet for transmission. AfterTcpDownShift returns, TcpSchedAckEmitted returns to its caller.

This processing has performed the function of selecting substitute sequence values for recycled acknowledgments in the process for establishing a limit on rate of emission of packets from the first transmission station to the second transmission station.

Direct feedback rate control effectively acts as a mechanism to feed back quality of service information to a TCP end system transmitter 12. Direct feedback rate

control does not need to store and forward `ACK only` packets, as these packets can be mechanically created, or even re-used.

In general, scheduling an exact target time for emission for an ACK only packet is non-problematic because the ACK packet is relatively small and is emitted into the flow opposite the prevailing data flow, which is less likely to be instantaneously congested.

Direct feedback rate control is effective for data flowing in either direction, and can even be applied independently to data flowing in either direction on the same connection. When forwarding TCP packets with data, the specific ACK sequence may be changed to control the transmission rate of the reciprocal direction of the connection.

Direct feedback rate control does not buffer or delay and TCP data, hence there is no system requirement for buffering. This helps make direct feedback rate control scaleable to large numbers of simultaneously managed connections.

Direct feedback rate control may increase or decrease the allotted "bandwidth" to any connection on the fly. When doing so, direct feedback rate control ensures that any decrease in transmission rate will be done smoothly and will not incur a retransmission timeout at the transmitter. Direct feedback rate control will also suppress retransmissions when it is known that their is a `delayed ack` pending for the retransmitted data.

Direct feedback rate control may pass through multiple equal ACK sequences to allow the `quick recovery` mechanism to work.

Direct feedback rate control allows for precise, explicit, bi-directional control of individual flows in terms of committed and excess information rate. Further, committed and excess information rate assignments may be set to scale to a given flow's potential speed, e.g. a T1 user in a given traffic class may be assigned a committed rate of 50 Kbps, where a dial-up user may receive a committed rate of 10 Kbps.

Direct feedback rate control does not build up deep transmission queues or toss packets. It delivers data smoothly and consistently. If a given flow is using bandwidth from the excess pool, that share of excess bandwidth may be reduced on the fly. Direct feedback rate control smoothly downshifts and upshifts flow rates in a manner which does not incur retransmission or delay in end systems.

Direct feedback rate control has additional beneficial characteristics. It smoothens peaks in demand bursts on short time scales. Because of the minimal transmission queue, it allows for delay bounded traffic and does not globally flatten the speed of all

connections, as would occur with a deep transmission queue. It allows for the explicit control of individual flows whether or not there is congestion, And it provides a capability to secure network bandwidth from malicious or ill behaved users, e.g. to `debounce` a remote browser's reload button hits.

Because direct feedback rate control controls inbound as well as outbound bandwidth, administrators can select simple policies such as `net browsing gets only excess bandwidth at low priority`.

Direct feedback rate control can be used for the following:

As a mechanism to enforce policies for bandwidth allocation of committed and excess information rate assignments.

As a mechanism to manage `inbound` as well as outbound network bandwidth over a given access link.

As a mechanism to assign explicit throughput rates to individual TCP connections.

As a unique mechanism for feeding back quality of service information to TCP end systems.

It should be noted that an inherent attribute of this invention is the ability to assign rates dynamically, although the control mechanisms for assigning rates are beyond the scope of this disclosure.

The invention has now been explained with reference to specific embodiments. Other embodiments will be apparent to one of ordinary skill in the art. It is therefore not intended that this invention be limited, except as indicated by the appended claims.